대 한 민 국 특 허 청
KOREAN INTELLECTUAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office.

출 원 번 호 : 10-2002-0046757
Application Number PATENT-2002-0046757

출 원 년 월 일 : 2002년 08월 08일
Date of Application AUG 08, 2002

출 원 인 : 삼성전자 주식회사
Applicant(s) SAMSUNG ELECTRONICS CO., LTD.

2002 년 10 월 10 일

특 허 청
COMMISSIONER

【서지사항】

【서류명】	특허출원서
【권리구분】	특허
【수신처】	특허청장
【참조번호】	0001
【제출일자】	2002.08.08
【국제특허분류】	G06F
【발명의 명칭】	디바이스 드라이버 제어 공통화 방법
【발명의 영문명칭】	COMMON CONTROL IMPLEMENT METHOD FOR DEVICE DRIVER
【출원인】	
【명칭】	삼성전자 주식회사
【출원인코드】	1-1998-104271-3
【대리인】	
【성명】	이건주
【대리인코드】	9-1998-000339-8
【포괄위임등록번호】	1999-006038-0
【발명자】	
【성명의 국문표기】	이형균
【성명의 영문표기】	LEE,Hyong Kyun
【주민등록번호】	700825-1226923
【우편번호】	442-739
【주소】	경기도 수원시 팔달구 영통동 황골마을주공1단지 108동 703호
【국적】	KR
【심사청구】	청구
【취지】	특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정에 의한 출원심사 를 청구합니다. 대리인 이건주 (인)
【수수료】	
【기본출원료】	20 면 29,000 원
【가산출원료】	26 면 26,000 원
【우선권주장료】	0 건 0 원
【심사청구료】	11 항 461,000 원
【합계】	516,000 원

【요약서】

【요약】

본 발명은, 디바이스 드라이버 제어 공통화 방법에 있어서, 디바이스 독립형 액세스(DIA)계층을 어플리케이션 계층과 디바이스 드라이버 계층에 사이에 위치시키며, 상기 DIA계층에 상위 어플리케이션 계층 및 디바이스 드라이버 계층에 대한 표준화 룰을 적용시키는 제1과정과, 상기 어플리케이션 계층 및 디바이스 드라이버 계층은 상기 DIA계층의 표준화된 룰을 통해 상기 디바이스 드라이버 계층 및 상위 어플리케이션 계층을 상호 액세스하는 제2과정으로 이루어진다.

【대표도】

도 4

【색인어】

디바이스 드라이버, 어플리케이션, DIA(Device Independent Access)계층, 디바이스 핸들러 ID

【명세서】

【발명의 명칭】

디바이스 드라이버 제어 공통화 방법{COMMON CONTROL IMPLEMENT METHOD FOR DEVICE DRIVER}

【도면의 간단한 설명】

도 1은 동일한 어플리케이션에서 하위의 동일한 기능을 하는 디바이스가 A에서 B로 변경된 경우를 보여주는 도면,

도 2는 동일한 디바이스 드라이버를 2개의 어플리케이션이 사용하는 경우를 보여주는 도면,

도 3은 본 발명의 실시 예에 따른 디바이스 드라이버 제어 공통화를 위한 개념도,

도 4는 본 발명의 실시 예에 따른 디바이스 드라이버 제어 공통화를 위한 액세스 개념도,

도 5는 디바이스 초기화를 위한 Dia_InitDevice라는 API가 호출된 후 그려지는 DCB(Device Control Block)의 연결 구조를 설명하기 위한 도면,

도 6은 레벨1 초기화 단계에서의 최종 DCB블록을 보여주는 도면,

도 7은 레벨2에 해당되는 포트가 4개 있는 HDLC 디바이스가 초기화되었을 때의 DCB의 연결 구조를 설명하기 위한 도면,

도 8은 채널이 초기화되었을 때의 DCB의 연결 구조를 설명하기 위한 도면,

도 9는 이벤트 테이블(Event Table) 구조를 설명하기 위한 도면,

도 10은 본 발명의 실시 예에 따라 디바이스 드라이버가 변경(교체)된 경우를 설명하기 위한 도면,

도 11은 종래 기술에서의 상위 어플리케이션(프로그램)이 변경된 경우를 설명하기 위한 도면,

도 12는 종래 기술에서의 하위 디바이스가 변경된 경우를 설명하기 위한 도면,

도 13은 본 발명의 실시 예에 따라 상위 어플리케이션(프로그램)이 변경된 경우를 설명하기 위한 도면,

도 14는 본 발명의 실시 예에 따라 하위 디바이스가 변경된 경우를 설명하기 위한 도면.

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<15> 본 발명은 통신 시스템에 관한 것으로, 특히 하드웨어 칩레벨 디바이스 드라이버의 제어를 공통화 하기 위한 방법에 관한 것이다.

<16> 일반적으로 상위 어플리케이션(application)을 만드는 사용자와 하위의 디바이스 드라이버(device driver)를 만드는 사용자간에는 대화형성 채널(communication channel)이 열려 있지 않고 있는 실정이다. 더군다나 요즈음에

는 많은 디바이스 드라이버들이 벤더(vendor)에 의해서 제공이 되어지고 있다. 이는 더욱 더 상위 어플리케이션 사용자와 하위 드라이버 사용자간의 대화형성 채널이 단절됨을 의미한다. 그에 따라 상위 어플리케이션은 상위대로, 하위 디바이스 드라이버는 하위대로 각각의 기준에 근거하여 API(Application Program Interface)를 작성하고 그 틀에 맞추어 상대가 사용하기를 바란다.

<17> 도 1은 동일한 어플리케이션에서 하위의 동일한 기능을 하는 디바이스가 A에서 B로 변경된 경우를 보여주고 있고, 도 2는 동일한 디바이스 드라이버를 2개의 어플리케이션이 사용하는 경우를 보여주고 있다.

<18> 먼저 도 1을 참조하면, 만일 동일한 어플리케이션(2)에서 하위 디바이스가 A에서 B로 변경된다면 이에 따른 디바이스 드라이버도 역시 디바이스A 드라이버(4)에서 디바이스B 드라이버(6)로 변경된다. 그렇게 되면 디바이스 드라이버 사용자는 자신의 디바이스 드라이버를 자신의 취향대로 만들게 되고 동일한 기능을 하는 API는 디바이스 A와 디바이스 B에 대해서는 전혀 다른 성격을 띄게 된다. 그에 따라 상위 어플리케이션(2)도 변경된 디바이스 및 디바이스 드라이버에 맞추어 변경을 해야 한다. 이렇게 변경이 되고 나면, 하위 디바이스 드라이버에 맞추어 제대로 작동을 하는지를 일일이 디바이스 드라이버의 모든 API에 대하여 검증을 새로 하여야 하며, 이에 관련된 모든 상위 어플리케이션도 마찬가지로 영향을 미치는 것이 있는지 확인을 하여야 한다.

<19> 다음으로 도 2를 참조하면, 상위 어플리케이션 A(10)와 B(12)는 동일한 디바이스 드라이버(14)를 사용한다. 디바이스 드라이버(14)가 동일한 기능을 하는 디바

이스 드라이버이기는 하지만 상위 어플리케이션에서는 드라이버로부터 얻고자 하는 정도가 서로 다를 수가 있다. 예를 들어, 어플리케이션 A(10)에서는 디바이스 드라이버(14)의 특정부분을 A-1, A-2라는 2개의 포맷으로 요구할 수가 있고 어플리케이션 B(12)에서는 디바이스 드라이버(14)의 상기 특정부분을 포맷 B-1, B-2, B-3의 3가지로 요구하는 경우가 발생할 수 있다. 이러한 경우는 어플리케이션 상호간의 협약이 맺어지지 않아서 발생을 하는 경우로서 디바이스 드라이버(14)에서는 그때마다 새로이 포맷을 변경을 해주어야 하며, API도 변경 혹은 추가 시켜주어야 한다. 상위 어플리케이션 역시도 하위 디바이스 드라이버가 자신이 원하는 포맷을 지원해주지 못하는 상황이 닥치면 자신의 어플리케이션 전체에 영향을 미치는 코드(code) 혹은 구조(structure)를 수정하여야 하는 문제가 야기될 수 있다. 이렇게 되면 상위 어플리케이션은 물론 기존의 검증이 끝났던 디바이스 드라이버도 또 다시 검증이 이루어져야 한다.

<20> 도 1 및 도 2를 참조하여 설명한 바와 같이 종래 기술은 동일한 어플리케이션에서 하위의 동일한 기능을 하는 디바이스가 A에서 B로 변경되거나, 동일한 디바이스 드라이버를 2개의 어플리케이션이 사용하는 경우의 특정 부분이 변경이 되면, 상위 어플리케이션 및 디바이스 드라이버에 대한 검증이 다시 이루어져야 한다. 이러한 검증 요구는 제품개발에 아주 큰 시간적, 물질적 악영향을 줄 수 있으며, 전체적으로 개발 시간(need time)의 증가로 제품개발의 경쟁력 저하를 가져올 수 있다.

【발명이 이루고자 하는 기술적 과제】

<21> 따라서 본 발명의 목적은 통신 시스템에 사용되는 상위 어플리케이션 계층과 하위 디바이스 드라이버 계층 사이에 DIA(Device Independent Access) 계층이라는 중간계층을

두어 상호 인터페이스를 맞추어 상위 어플리케이션 계층과 하위 디바이스 드라이버 계층이 독립적이고 공통화 되어 사용될 수 있도록 하는 방법을 제공하는데 있다.

<22> 본 발명의 다른 목적은 통신 시스템에 사용되는 하위 디바이스 드라이버 계층에 독립적이며 공통화 되어 사용될 수 있도록 하는 방법을 제공하는데 있다.

<23> 본 발명의 또 다른 목적은 통신 시스템에 사용되는 상위 어플리케이션 계층에 독립적이며 공통화 되어 사용될 수 있도록 하는 방법을 제공하는데 있다.

<24> 상기한 목적에 따라, 본 발명은, 디바이스 드라이버 제어 공통화 방법에 있어서, 디바이스 독립형 액세스(DIA)계층을 어플리케이션 계층과 디바이스 드라이버 계층에 사이에 위치시키며, 상기 DIA계층에 상위 어플리케이션 계층 및 디바이스 드라이버 계층에 대한 표준화 룰을 적용시키는 제1과정과, 상기 어플리케이션 계층 및 디바이스 드라이버 계층은 상기 DIA계층의 표준화된 룰을 통해 상기 디바이스 드라이버 계층 및 상위 어플리케이션 계층을 상호 액세스하는 제2과정으로 이루어짐을 특징으로 한다.

【발명의 구성 및 작용】

<25> 본 발명은, 통신 시스템에서 DIA(Device Independent Access)계층을 상위 어플리케이션 계층과 디바이스 드라이버 계층 사이에 위치시켜 상위 어플리케이션은 디바이스 드라이버가 아닌 DIA를, 역시 디바이스 드라이버는 상위 어플리케이션이 아닌 DIA의 표준룰(rule)을 통해 상호 액세스하므로 직접적인 액세스를 피하게 한다. 이렇게 함으로써, 상위 어플리케이션 및 하위의 디바이스 드라이버가 개별화되고 특화되는 성향을 배제시

키고 표준화 룰을 적용하여 개발의 요구 시간(need time)을 줄일 뿐 아니라 개발비용의 절감, 개발의 신뢰성을 크게 향상시킬 수 있다.

<26> 이하 본 발명의 바람직한 실시 예들을 첨부한 도면을 참조하여 상세히 설명한다. 도면들 중 동일한 구성요소들은 가능한 한 어느 곳에서든지 동일한 부호들로 나타내고 있음에 유의해야 한다. 또한 본 발명의 요지를 불필요하게 흐릴 수 있는 공지 기능 및 구성에 대한 상세한 설명은 생략한다.

<27> 도 3은 본 발명의 실시 예에 따른 디바이스 드라이버 제어 공통화를 위한 개념 구성도이다. 도 3을 참조하면, 본 발명의 실시 예에서는 통신 시스템에서 DIA(Device Independent Access)계층(22)을 상위 어플리케이션(20)과 디바이스 드라이버(24,26) 사이에 위치시킨다. 그래서 상위 어플리케이션(20)은 디바이스 드라이버(24,26)가 아닌 DIA계층(22)의 표준 룰을 통해, 마찬가지로 디바이스 드라이버(24,26)는 상위 어플리케이션(20)이 아닌 DIA계층(22)의 표준 룰(rule)을 통해 상호 액세스한다. 이를 도 4를 참조하여 더욱 상세히 설명하면 하기와 같다.

<28> 도 4는 본 발명의 실시 예에 따른 디바이스 드라이버 제어 공통화를 위한 액세스 개념도이다. 도 4를 참조하면, 상위 어플리케이션(20)은 표준화된 공통포맷으로 코멘드 예컨대, LOS(loss of state)상태를 DIA계층(22)으로 요청을 하게 되면, DIA계층(22)은 상기 요청을 디바이스A 로컬포맷 형태로 변환하여 디바이스A 드라이버(24)로 LOS상태를 요청한다. 그에 따라 디바이스A 드라이버(24)는 DIA계층(22)에게 디바이스A 로컬포맷 형태로 LOS상태 응답을 하게 되고, DIA계층(22)은 상기 응답을 표준화된 공통포맷으로 변환하여 LOS(loss of state)상태를 상위 어플리케이션(20)으로 응답하게 된다. 한편 디바이스가 A에서 B로 변경이 되게 되면 디바이스 드라이버도 디바이스A 드라이버(24)에서

디바이스B 드라이버(26)로 변경이 된다. 상위 어플리케이션(20)은 디바이스 변경에 상관없이 표준화된 공통포맷으로 코멘드 예컨대, LOS(loss of state)상태를 DIA계층(22)으로 요청하며, DIA계층(22)은 상기 요청을 디바이스B 로컬포맷 형태로 변환하여 디바이스B 드라이버(26)로 LOS상태를 요청한다. 그에 따라 디바이스B 드라이버(26)는 DIA계층(22)에게 디바이스B 로컬포맷 형태로 LOS상태 응답을 하게 되고, DIA계층(22)은 상기 응답을 표준화된 공통포맷으로 변환하여 LOS(loss of state)상태를 상위 어플리케이션(20)으로 응답하게 된다. 상기와 같이, 상위 어플리케이션(20)과 하위 디바이스 드라이버(24,26) 사이에 존재하는 DIA계층(22)이 표준 룰을 가지고 상호 인터페이스를 맞추어 주므로 어플리케이션 변경이나 디바이스 변경에 따른 상위 어플리케이션 또는 디바이스 드라이버에 대한 검증이 필요가 없다.

<29> 표준 룰을 가지고 상위 어플리케이션(20)과 하위 디바이스 드라이버(24,26)간의 상호 인터페이스를 맞추어 주기 위해, 본 발명의 실시 예에 따른 DIA 계층(22)은 디바이스 드라이버 제어 블록(Device Driver Control Block: 이하 "DDCB"라 칭함)에서 자료를 읽어 DIA계층(22)에 의해 표준화된 룰에 정의된 기능들을 이용하여 디바이스 드라이버를 액세스한다. 상기 DDCB는 디바이스 드라이버들 각각을 공통화하기 위해 정의한 것으로 해당 기능의 존재 여부 및 위치를 알려주는 블록이다.

<30> 그리고 본 발명의 실시 예에서는 ITU/RFC 등의 국제 기구에서 기 정의된 표준화된 펑션블록(function block)의 모든 펑션들중 해당 디바이스 드라이버에서 사용 가능한 부분만을 펑션 테이블로서 정의한다. 대부분의 통신 시스템은 ITU(International Telecommunications Union), IETF(Internet Engineering Task Force), ETSI(European Telecommunications Standardization Institute), ATM 포럼(Asynchronous Transfer Mode

forum), ADSL 포럼(Asymmetrical Digital Subscriber Line forum) 등과 같은 국제 표준 기구에서 제정된 표준화된 문서에 정의되어진 모든 펑션 블록(function block)들이 본 발명에서 사용될 수 있다. 본 발명의 실시 예에서는 국제 표준 기구에서 제정된 표준화된 문서에 정의된 모든 펑션 블록(function block)들에서 해당 디바이스 드라이버가 사용 가능한 부분만을 다시 정의하여 펑션테이블로서 사용한다. 본 발명의 실시 예에 따라 사용되는 펑션 테이블은 표준화된 코멘드 제어 테이블(command control table)로서 메모리 영역에 저장된다.

<31> 또한 본 발명의 실시 예에서는 상위 어플리케이션 개발자들이 디바이스를 쉽게 제어하기 위해서, DIA계층(22)에서 정해 놓은 표준 데이터 포맷인 디바이스 핸들러 ID를 사용한다. 상기 디바이스 핸들러 ID는 본 발명의 실시 예에 따른 표준화된 데이터 포맷을 가지며 디바이스들 각각에 대한 고유 식별자이다. 상기 디바이스 핸들러ID는 해당 디바이스의 초기화중에 DIA계층(22)에 의해서 상위 어플리케이션(20)으로 제공된다. 상위 어플리케이션(20)은 해당 디바이스의 디바이스 핸들러ID를 저장하고 있다가, 해당 디바이스 호출이 필요할 경우에 상기 디바이스 핸들러ID를 이용하여 호출을 한다. 그에 따라 DIA계층(22)에서는 상기 디바이스 핸들러ID에 의거하여 어느 디바이스의 드라이버를 호출해야하는지를 결정하고 결정한 디바이스 드라이버를 호출한다.

<32> 상기 DIA 계층(22)에 의해 생성되어 상위 어플리케이션(20)으로 제공되는 표준화된 고유 식별자인 디바이스 핸들러ID 및 표준화된 코멘드 제어 테이블(command control table)에 대해서 도 5 내지 도 8을 참조하여 보다 상세히 설명한다. 또한 본 발명의 실시 예에서 언급되는 해당 디바이스 이벤트 테이블(device event table), 해당 디바이스

모듈의 프로파일(profile)에 대해서도 도 5 내지 도 8을 참조하여 보다 상세히 설명한다.

<33> 도 5는 디바이스 초기화를 위한 Dia_InitDevice라는 DIA계층(22)의 API가 어플리케이션 사용자에 의해 호출된 후 그려지는 DCB(Device Control Block)의 연결을 설명하기 위한 도면이고, 도 6은 레벨1 초기화 단계에서의 최종 DCB블록을 보여주는 도면이며, 도 7은 레벨2가 4개 있는 디바이스가 초기화되었을 때의 DCB의 연결 구조를 설명하기 위한 도면이다. 그리고 도 8은 채널 초기화되었을 때의 DCB의 연결 구조를 설명하기 위한 도면이다.

<34> 먼저 디바이스 핸들러ID에 대해서 상세히 설명하면 하기와 같다.

<35> DIA계층(22)에 의해 생성되는 해당 디바이스의 디바이스 핸들러ID는 도 5에 도시된 DCB handlerId[1.0.0]과 같은 일 예로서 표현된다. 상기 디바이스 핸들러ID는 레벨1, 레벨2, 채널 3가지의 비부호 정수(Unsigned Integer)의 구조로 구성되며, 고유한(unique) 값을 시스템내에서 갖는다. 디바이스 핸들러 ID는 x1.x2.x3(x는 정수)으로 구성되어 있다. 디바이스 핸들러 ID는 DCB handlerId[x1.x2.x3]로서 표현된다. 여기서, x1은 디바이스ID를 의미하는 레벨1의 값이고, x2는 해당 디바이스가 가지고 있는 논리적 혹은 물리적 그룹 번호를 의미하는 레벨2의 값이며, x3은 해당 디바이스 혹은 그룹이 가지고 있는 채널번호를 의미하는 채널값이다. x1,x2,x3 값이 "0"이면, 이는 해당 레벨 혹은 채널이 존재하지 않음을 의미하는 것이다. 초기화(Initial)가 되어지면 도 6 내지 도 8에 도시된 일 예와 같이, 정수 x값은 "1"부터 순차적으로 증가를 한다. 레벨1의 값 즉 x1이 "0"인 핸들러는 존재할 수 없다. 이는 초기화(Initial)가 되어 있지 않음을 의미한다.

<36> 도 5에 도시된 바와 같이 레벨1이 초기화되면 본 발명의 실시 예에 따른 표준화 룰에 의거해 디바이스에 따라 호출되는 엘리먼트들을 담는 DCB(Device Control Block)(32)가 동적 할당된다. 상기 엘리멘트들은 DIA계층(22)에서의 표준화 룰을 수행하기 위한 다양한 포인터 및 펑션 포인터들로 구성되어 있다. 레벨1 초기화시 DCB(32)의 엘리먼트는, *pHandler, *fpIntivDevice, *fpOpenChannel, *fpCloseChannel, *fpRead, *fpWrite, *fpReset, *pControlTable, *pDDCB, *pEventTable, *pAnchor가 포함되어 있다. *pHandler는 디바이스 초기화시 주어지는 초기화 프로파일의 위치를 알려주는 포인터이고, *fpIntivDevice는 디바이스 초기화시 사용되는 펑션이 있는 펑션 포인터이다. *fpOpenChannel은 채널을 열 때 사용되는 펑션이 있는 펑션 포인터이고, *fpCloseChannel는 채널을 닫을 때 사용되는 펑션이 있는 펑션 포인터이며, *fpRead는 열린 채널의 데이터를 읽을 때 사용되는 펑션 포인터이고, *fpWrite는 열린 채널의 데이터를 쓸대 사용되는 펑션 포인터이다. *fpReset는 디바이스 리셋시의 펑션 포인터이고, *pControlTable은 코멘드 제어 테이블의 위치를 알려주는 포인터이며, *pDDCB는 디바이스 드라이버 제어 테이블(device driver control table)(36)이 있는 위치를 알려주는 포인터이다. *pEventTable은 이벤트 테이블(38)이 있는 위치를 알려주는 포인터이고, *pAnchor는 다음 레벨을 지시하는 포인터이다.

<37> 모든 디바이스 초기화가 수행된 후 DIA계층(22)은 생성된 디바이스 핸들러 ID를 상위 어플리케이션(20)으로 제공한다. 상위 어플리케이션(20)은 해당 디바이스를 초기화하고 부여받은 디바이스 핸들러 ID만을 저장하고 있다가 디바이스 핸들러 호출을 하면 DIA계층(22)은 이를 받아 들여 어느 디바이스의 드라이버를 호출해야 하는지를 결정하고 결정한 디바이스 드라이버를 호출하게 된다.

<38> 다음으로 펑션 테이블인 표준화된 코멘드 제어 테이블에 대해서 설명하면 하기와 같다.

<39> 표준화된 코멘드 제어 테이블은 해당 기능군 별로 ITU(International Telecommunications Union), IETF(Internet Engineering Task Force), ETSI(European Telecommunications Standardization Institute), ATM 포럼(Asynchronous Transfer Mode forum), ADSL 포럼(Asymmetrical Digital Subscriber Line forum) 등과 같은 국제 표준 기구에서 제정된 표준화된 문서의 펑션 블록(function block)과 해당 블록에서 언급하는 엘리먼트(element)들을 사용하여 제정되었으며, 해당분야에 추가적인 요구가 있을 시는 언제든지 추가가 가능하다. 본 발명의 발명자는 일 예로 초고속통신에 관련된 사항들을 테이블로 작성하였다. 작성된 테이블들은 해당 메모리 영역에 저장된다. 그리고 기능에 대한 추가가 필요시에는 즉시 반영을 할 수 있음을 유의하여야한다.

<40> 도 5에 도시된 바와 같이, 표준화된 코멘드 제어 테이블(34)은 해당 DCB의 *pControlTable 포인터에 의해서 지시되는 메모리 영역에 위치된다. 상기 코멘드 제어테이블(34)의 코멘드ID commandID는 표준화되어 고유한(unique) 값으로 정해져 있다. commandID에는 코멘드 펑션의 펑션 포인트 *fpCommandFn이 맵핑되어 있다.

<41> 하기 표 1은 본 발명의 실시 예에 따른 코멘드 제어 테이블(34)의 구조체(structure)이다.

<42>

【표 1】

<<표준화된 코멘드 제어 테이블의 구조체>>

```
typedef enum
{
        D_SDH_COMMAND_ID_START = 0x2000000,
        D_SDH_SPI_COMMAND_ID_START = 0x2000100,
        D_SDH_SPI_SET_ALS,
        D_SDH_SPI_GET_ALS,
        ....
        D_SDH_COMMAND_ID_END
} DIA_COMMAND_ID_SDH_E;
```

<43> 표 1에 기재된 'DIA_COMMAND_ID_SDH_E'와 같은 표준 코멘드 typedef enum(enumulate)이, 각 분야별로는 DIA_COMMAND_ID_PDH_E, DIA_COMMAND_ID_ATM_E 등과 같이 존재하며, 기능 추가가 자유롭다. 이 표준화된 코멘드 제어 테이블(Control Table)(34)을 통해 상위 어플리케이션 사용자는 언제든지 해당 디바이스 드라이버를 호출할 수 있다. 디바이스 드라이버가 변경되어도 해당 표준 코멘드는 동일하게 존재한다. 단지 해당 코멘드가 실제 해당 디바이스 드라이버(Device Driver)에 의해서 수행되어지는 평션의 포인터만 달라진다. 평션의 포인터는 디바이스 초기화시에 디바이스 드라이버에 의해서 DIA계층(22)으로 제공된다. 상기 코멘드는 어플리케이션 사용자에 의해 Dia_Control이라는 API를 통해서 호출되어진다. 실제 상위 어플리케이션 사용자가 상기 Dia_Control이라는 API를 통해 특정 제어 코멘드를 호출할 시 DIA계층(22) 하부에 있는 디바이스 드라이버만이 달라지는 것이며 상위의 어플리케이션(20)의 사용자는 동일한 코

멘드를 동일한 형태로 호출하기 때문에 전혀 문제를 야기하지 않고, 전혀 수정을 하지 않고 사용을 한다.

<44> 다음으로 본 발명의 실시 예에 따른 디바이스 드라이버 제어 테이블(Device Driver Control Table)(36)에 대해서 도 5를 참조하여 설명한다. 도 5에서, 디바이스 드라이버 제어 테이블(Device Driver Control Table)(36)은 *pDDCB 포인터에 의해 지시하는 테이블로서, InitLevel1, InitLevel2, OpenChannel, CloseChannel, 제어 테이블, 이벤트 테이블 등의 위치를 지시하는 펑션 포인트(function point) *fp 예컨대, *fpInitLevel1, *fpInitLevel2, *fpOpenChannel, *fpCloseChannel, *fpControlTable(미도시됨), *fpEventTable(미도시됨) 등을 가지고 있다. 또한 디바이스에 고유한 초기화 펑션 수, 포트 수, 및 채널 수를 나타내는 정보(미도시됨)를 가진다. 상기 디바이스 드라이버 제어 테이블(36)은 디바이스 드라이버들 각각을 공통화하기 위해 정의되며 해당 펑션의 존재 여부 및 위치를 알려주는 블록인 DDCB(Device Driver Control Block)에 해당된다. 디바이스 드라이버 제어 테이블(36)은 *fpInitLevel1, *fpInitLevel2, *fpOpenChannel, *fpCloseChannel, *fpControlTable(미도시됨), *fpEventTable(미도시됨) 등을 가지고 있다. 또한 디바이스에 고유한 초기화 펑션 수, 포트 수, 및 채널 수를 나타내는 정보(미도시됨)를 가진다. 디바이스 초기화시 *pDDCB 포인터에 의해서 DDCB인 디바이스 드라이버 제어 테이블(36)을 찾아가며, 그 자료들 *fpInitLevel1, *fpInitLevel2, *fpOpenChannel, *fpCloseChannel, *fpControlTable(미도시됨), *fpEventTable(미도시됨)등을 이용해서 할당된 DCB(32)에 정보로서 채워진다.

<45> 다음으로 본 발명의 실시 예에 관련된 이벤트 테이블에 대해서 설명하면 하기와 같다.

<46> 도 5에 도시된 이벤트 테이블(Event Table)(38)은 해당 DCB의 *pEventTable 포인터가 지시하는 곳에 있는 구조체(structure)로서 필요시에만 생성된다. 이벤트를 사용하지 않을 시는 "Null"로 지시된다. 이벤트 테이블(38)은 이벤트ID "EventId"와 이벤트 리스트 구조체 포인터(event list structure pointer) "*pEventList"로 되어 있으며, 각각의 이벤트 리스트는 도 9에 도시된 바와 같이, 링크드 리스트(linked list) 구조로 되어 있다. 그래서 하나의 이벤트에 여러 곳으로의 통지(notification) 혹은 콜백 펑션(call-back function) "Call-back Fn"을 호출할 수 있는 구조이다. 초기 이벤트 테이블(38)에는 "EventId"만 존재하고 "*pEventList"는 최초 "Null"로 지시된다. 상위 어플리케이션(20)의 사용자가 해당 이벤트를 사용하고자 할 시에는 "Dia_Register"라는 API를 이용하여 "*pEventList"에 이벤트 리스트 구조체(event list structure)의 포인터를 연결함으로써 등록 사용할 수 있다.

<47> 하기 표 2는 본 발명의 실시 예에 따른 이벤트 테이블(38)의 구조체(structure) 일예 구성도이다.

<48>

【표 2】

```
typedef struct
{
        EXECFUNC          pCallBackFunc;
        EXACTLY_INT32_T   *pNextCallback;
}DIA_EVENT_CONFIG_T;

typedef struct
{
    EXACTLY_UINT32_T        eventId;
    DIA_EVENT_CONFIG_T      *pEventList;
}DIA_EVENT_TABLE_T;
```

<49> 다음으로 본 발명의 실시 예에 관련된 디바이스 모듈의 프로파일(profile)에 대해서 설명하면 하기와 같다.

<50> 도 5에 도시된, 해당 DCB(32)의 *fpInitDevice 평션 실행 시 필요할 수 있는 입력 독립변수(input argument)로서 구조체 포인터(structure pointer)의 형태를 띌 수 있다. 특별한 경우에는 해당 디바이스의 구조체(structure)가 정의되어 질 수도 있지만 통상의 경우는, 레벨1 초기화 시에 기본 어드레스(base address)가, 레벨2 는 그룹ID(group Id)가, 채널 초기화 시에 채널ID(Channel Id)가 들어가는 공통 구조체(common structure)가 사용되어진다.

<51> 하기 표 3은 본 발명의 실시 예에 따른 디바이스 모듈의 프로파일 공통 구조체(structure) 일예 구성도이다.

<52> 【표 3】

```
typedef struct
{
        EXACTLY_UINT32_T  baseAddress1;
        EXACTLY_UINT32_T  numExtraBaseAddresses;
        EXACTLY_UINT32_T  *pExtraBaseAddresses;
} DIA_BASE_ADDR_T;

typedef struct
{
        DIA_BASE_ADDR_T   baseAddress;
        EXACTLY_UINT32_T  *pUserDefine;
} DIA_COMMON_LEVEL1_PROFILE_T;

typedef struct
{
        EXACTLY_INT32_T   groupNo;
        EXACTLY_UINT32_T  *pUserDefine;
} DIA_COMMON_LEVEL2_PROFILE_T;

typedef struct
{
        EXACTLY_INT32_T   channelNo;
        EXACTLY_UINT32_T  *pUserDefine;
} DIA_COMMON_CHANNEL_PROFILE_T;
```

<53> 이하 본 발명의 실시 예에 따른 동작을 상세히 설명하면 하기와 같다.

<54> 상위 어플리케이션(20)에서는 사용되어질 펑션블록의 펑션을 호출하게 되며, DIA계층(22)에서는 해당 펑션테이블들의 존재 유무 및 위치를 지시해 주는 DDCB(Device Driver Control Block)를 이용하여 해당 디바이스 드라이버의 펑션 테이블로부터 해당 펑션의 존재 유무를 찾아낸다. 만약 존재하게 되면 해당 펑션을 호출한다. 이를 가능케 하기 위해서 DIA계층(22)은 해당 디바이스 초기화 시에 디바이스 핸들러 ID를 상위 어플

리케이션(20)으로 알려주고, 상위 어플리케이션은 이를 이용하여 하위 디바이스 드라이버를 직접 액세스하게 된다.

<55> 먼저 본 발명의 실시 예에 따른 디바이스 초기화시의 동작을 설명하면 하기와 같다.

<56> 상위 어플리케이션 사용자가 디바이스 초기화를 위해 Dia_InitDevice라는 API를 호출을 하게 되며, DIA계층(22)은 해당 펑션테이블들의 위치를 지시해 주는 DDCB(Device Driver Control Block)를 이용하여 상위 어플리케이션(20)에 의해서 사용되어질 펑션블록들(예컨대, 제어 테이블, 이벤트 테이블 등등) 각각에 대한 포인터들을 모아놓은 DCB(32)를 동적으로 할당한다.

<57> 하기에서는 Dia_InitDevice가 호출(call)된 후에 그려지는 DCB(Device Control Block) 블록 데이터베이스의 연결이 도 5 내지 도 8이 참조되어 상세히 설명될 것이다. 하기에서는 기본 흐름의 형태를 따라가면서, 데이터베이스의 구조를 추가해 가는 방식을 취하겠다.

<58> (1) 레벨1 초기화

<59> 초기화하고자 하는 디바이스는 레벨1 혹은 레벨2, 채널을 가질 수 있다. 레벨1 초기화 시에는 해당 디바이스의 최상위 블록인 DCB(Device Control Block)(32)가 동적 할당되며, 이후에 해당 디바이스의 레벨2는 레벨1 DCB(32)의 앵커(anchor)(앵커 포인터 *pAnchor에 의해서 지시됨)를 통해서 참조되도록 한다. 각 카드에 사용되는 디바이스의 개수는 이미 알고 있는 값이므로, 처음에 글로벌 변수(global variable)로서 도 5에 도

시된 디바이스ID deviceId와 DCB(32)를 가리키는 포인터 *pDCB를 가지는 최상위 데이터베이스(40)는 정의되어 있다고 가정한다.

<60> 레벨 1 초기화되었을 때의 형태는 도 5와 같다. 그리고 레벨1 초기화 단계에서의 최종 DCB 블록을 도시한 것이 도 6이다. 도 5와 같은 DCB(32)를 생성할 때마다 DIA계층(22)은 도 6에 도시된 바와 같이, 해당 DCB마다 고유한 디바이스 핸들러ID값 DCB HandlerId[x1.x2.x3]를 상위 어플리케이션(20)으로 리턴한다. 즉 DIA계층(22)에서는 디바이스의 레벨1 초기화 순서에 따라 상기 x1값을 "1"부터 순차적으로 증가시키면서 디바이스 핸들러ID값 DCB HandlerId[x1.x2.x3]을 부여한다. 도 6의 일 예를 참조하면, HDLC(High level Data Link Control)의 디바이스 핸들러ID값은 DCB HandlerId[1.0.0]로서 리턴되며, LAN(Local Area Network)의 디바이스 핸들러ID값은 DCB HandlerId[2.0.0]로서 리턴되며, SERIAL의 디바이스 핸들러ID값은 DCB HandlerId[3.0.0]로서 리턴되며, UTOPIA(Universal Test & Operations Physical layer Interface for ATM)의 디바이스 핸들러ID값은 DCB HandlerId[4.0.0]로서 리턴된다.

<61> 상위 어플리케이션(20)은 리턴되는 상기 디바이스 핸들러ID값 DCB HandlerId[x1.x2.x3]을 관리해야 한다. 상위 어플리케이션(20)의 사용자는 리턴되는 디바이스 핸들러ID값 DCB HandlerId[x1.x2.x3]를 이용해서 해당하는 기능의 디바이스 펑션 호출(device function call)을 할 수 있다. 또한 디바이스 초기화 순서에 따라서 디바이스에 해당되는 디바이스 핸들러ID가 부여되므로 상위 사용자는 리턴되는 디바이스 핸들러ID가 어떤 디바이스에 해당하는지 관리해야 한다. 예컨대, 3개의 HDLC1,HDLC2,HDLC3의 디바이스가 있는데, HDLC3이 먼저 초기화되었다면 HDLC3 디바이스의 x1값이 "1"이 된다.

<62> (2) 레벨2 초기화

<63> DIA계층(22)은 레벨1 초기화 후에 레벨2 초기화를 할 수 있다. 이때 사용되는 정보는 DDCB(Device Driver Control Block: 디바이스에 고유한 초기화 펑션(init function) 및 포트 수, 채널 수를 가짐)의 레벨2에서의 허용 개수(예를 들면 포트 수)이다. 예컨대 해당 디바이스의 포트의 수를 참조하여 레벨2 초기화를 할 수 있도록 앵커(anchor)를 할당한다. 그리고 DIA계층(22)은 레벨2에 필요한 DCB 블록을 참조할 수 있도록 레벨1의 앵커에 생성된 DCB(32)의 어드레스를 지정한다.

<64> 도 7은 레벨2에 해당되는 포트가 4개 있는 HDLC 디바이스가 초기화되었을 때의 도면이다. 도 7을 참조하면, 4개의 포트들 각각에 대응되어 할당된 앵커 Anchor1, Anchor2, Anchor3, Anchor4는 디바이스 핸들러ID값이 각각 DCB HandlerId[1.1.0], DCB HandlerId[1.2.0], DCBHandlerId[1.3.0], DCB HandlerId[1.4.0]로서 리턴된다. 예컨대, 4개의 포트가 포트0,포트1,포트2,포트3으로 되어 있으면, 포트0이 앵커 Anchor1에, 포트1이 앵커 Anchor2에, 포트2가 앵커 Anchor3에, 포트3이 앵커 Anchor4에 할당되며, 디바이스 핸들러ID값이 각각 DCB HandlerId[1.1.0], DCB HandlerId[1.2.0], DCBHandlerId[1.3.0], DCB HandlerId[1.4.0]로서 부여될 수 있다.

<65> (3) 채널 초기화

<66> 채널의 DCB 생성은 필요시에만 하도록 한다. 레벨2가 없이 채널만 있는 경우에는 도 8에 도시된 바와 같이 앵커(anchor)의 0번 Anchor0에 DCB를 연결한다.

<67> 도 8에 도시된 채널 초기화의 DCB 연결 구조는 도 7과 유사하게 채널의 수만큼 앵커를 할당하고, 이 앵커에 DCB 블록이 연결된다. 이때도 역시, 디바이스 핸들러ID값은 리턴된다. 만일 앵커 Anchor1에 대응된 포트에 4개의 채널(채널1, 채널2, 채널3, 채널4)이 있는데 그중 채널3을 먼저 오픈(open)한다면, 초기화(initial) 순서에 따라서 먼저 오픈된 채널3에 대해서 핸들러ID DCB HandlerId[2.1.1]값을 리턴한다. 그러므로, 상위 어플리케이션 사용자는 전기 (1)절에 언급했듯이 디바이스 초기화 순서에 따라서 디바이스에 해당되는 디바이스 핸들러ID가 부여되므로 해당 채널과 그에 대응된 디바이스 핸들러ID 값의 맵핑을 관리해야 한다.

<68> 디바이스 초기화 후 상위 어플리케이션(20)은 표준화된 식별자인 디바이스 핸들러ID값 DCB HandlerId[x1.x2.x3]를 이용해서 사용되어질 펑션블록의 펑션을 호출한다. 그에 따라 DIA계층(22)에서는 해당 펑션테이블들의 위치를 지시해 주는 DDCB(Device Driver Control Block)를 이용하여 해당 디바이스 드라이버의 펑션 테이블로부터 해당 펑션의 존재 유무를 찾아낸다. 만약 존재하게 되면 해당 펑션을 호출한다.

<69> 도 10은 본 발명의 실시 예에 따라 디바이스 드라이버가 변경(교체)된 경우를 설명하기 위한 도면이다.

<70> 도 10을 참조하면, 본 발명의 실시 예에 따른 DIA계층(22)에 의해서 디바이스 드라이버가 A에서 B로 변경이 되더라도, 디바이스 HDLC에 대한 디바이스 핸들러ID값 DCB handlerId[1.0.0]는 전혀 변화가 없다. 변화가 되는 것은 디바이스 초기화시 DIA계층(22)의 제어에 의해 하위 디바이스 드라이버와 DCB(32)의 포인터(어드레스)들만이 새로운 디바이스의 것들을 지시하게 된다. 그에 따라 상위 어플리케이션(20)은 하위의 디바

이스 드라이버 A에서 B로 변경되었더라도 전혀 변경 없이 동일한 기능을 사용할 수 있다.

<71> 도 11 내지 도 14는 종래 기술과 본 발명의 실시 예에 따라 상위 어플리케이션이나 하위 디바이스가 변경되는 경우에 수행되어져야 하는 작업을 비교하기 위한 도면들이다.

<72> 도 11 및 도 13은 상위 어플리케이션(프로그램)이 변경된 경우를 예로 든 것이다. 도 11은 종래 기술에서의 상위 어플리케이션(프로그램)이 변경된 경우를, 도 13은 본 발명의 실시 예에 따라 상위 어플리케이션(프로그램)이 변경된 경우를 보여주고 있다.

<73> 먼저 도 11을 참조하면, 종래기술에서는 상위 어플리케이션 A에서 어플리케이션 B로 변경되면, 상위 어플리케이션만이 변경되는 것이 아니라, 이에 해당하는 하위의 디바이스 드라이버도 마찬가지로 변경되어야 한다. 그 이유는, 어플리케이션 B에서 사용되어지는 구조(structure)들과 API들이 어플리케이션 A의 그것들과는 전혀 상이하므로 하위 디바이스 드라이버에게 요청되어지는 API들의 형식이 바뀌게 된다. 그러므로 하위 디바이스 드라이버는 기존의 것에서 추가, 삭제, 수정의 작업이 다시 이루어지게 되며 이로 인하여 이 부분들에 대하여 검증작업을 새로이 이루어져야 한다. 물론, 이를 이용하여야 하는 상위 어플리케이션 B에 대한 검증 작업도 함께 이루어져한다.

<74> 하지만 도 13에 도시된 바와 같이 본 발명의 실시 예에서는 도 11과 함께 설명한 종래기술의 단점을 보완하기 위하여 상위 어플리케이션 계층과 하위 디바이스 드라이버 계층 사이에 DIA 계층(22)을 두고 있다. 상위 어플리케이션 계층과 하위 디바이스 드라이버 계층 사이에 DIA 계층(22)을 두게 되면 상위 어플리케이션이 A에서 B로 완전히 바뀌어도 상위 어플리케이션 사용자는 DIA 표준 룰을 따르므로 어플리케이션 A와 어플리케

이션 B가 사용하는 제어 코멘드(control command) Control A, Control B, Control C는 항상 동일하다. 상위 어플리케이션은 직접적으로 디바이스 드라이버를 제어하는 것이 아니라 디바이스가 가지고 있는 제어코멘드를 DIA 계층(22)을 통해 제어하게 된다. 도 13을 일 예로 설명하면, 어플리케이션 A와 어플리케이션 B가 사용하는 표준화된 공통 포맷의 제어 코멘드(control command) Control A, Control B, Control C는 DIA계층(22)에 의해서 디바이스1이 가지고 있는 제어 코멘드 Control A-1, Control B-1, Control C-1로 개별 변환이 되고, 변환된 제어 코멘드 Control A-1, Control B-1, Control C-1이 디바이스1 드라이버에 제공된다. 그러므로, 상위 어플리케이션은 하위 디바이스 드라이버를 신경 쓰지 않아도 된다.

<75> 도 12와 도 14는 상위 어플리케이션은 동일하고 동일한 기능을 하는 하위 디바이스가 변경되어 각각의 디바이스 드라이버가 여러 개 존재하는 경우를 표현한 것이다. 도 12는 종래 기술에서의 하위 디바이스가 변경된 경우를, 도 14는 본 발명의 실시 예에 따라 하위 디바이스가 변경된 경우를 보여주고 있다.

<76> 먼저 도 12를 참조하면, 종래 기술에서는 상위 어플리케이션은 그대로이고 하위 디바이스 드라이버가 변경되면, 하위 디바이스 드라이버의 API가 바뀌는 상황이므로 이 디바이스 드라이버의 API를 사용하는 상위 어플리케이션도 마찬가지로 추가, 삭제 혹은 수정이 되어야 한다. 그렇게 되면 하위 바뀐 디바이스 드라이버뿐만 아니라 상위 어플리케이션도 다시 검증을 해야 한다.

<77> 하지만 도 14에 도시된 바와 같이 본 발명의 실시 예에 따라 상위 어플리케이션 계층과 하위 디바이스 드라이버 계층 사이에 DIA 계층(22)을 두게 되면, 상위 어플리케이션 계층은 전혀 변화가 없고 단지 하위 디바이스 드라이버만이 그의 상위 계층인 DIA

계층(22)에 맞추어 주면 모든 작업이 끝이 난다. 이럴 경우는 하위 디바이스 드라이버의 검증만으로 모든 작업이 끝날 수 있다. 도 14를 일 예로 설명하면, 어플리케이션 A가 사용하는 표준화된 공통 포맷의 제어 코멘드(control command) Control A, Control B, Control C는 DIA계층(22)에 의해서 디바이스1이 가지고 있는 제어코멘드 Control A-1, Control B-1, Control C-1로 개별 변환이 되고, 변환된 제어코멘드 Control A-1, Control B-1, Control C-1이 디바이스1 드라이버에 제공된다. 그런데, 하위 디바이스 드라이버가 디바이스1 드라이버에서 디바이스2 드라이버로 변경이 되면, 하위 디바이스2 드라이버를 디바이스2가 사용하는 제어코멘드가 Control A-2, Control B-2, Control C-2임을 상위 계층인 DIA계층(22)에 알려서, DIA계층(22)이 어플리케이션 A가 사용하는 표준화된 공통 포맷의 제어 코멘드(control command) Control A, Control B, Control C를 디바이스2가 가지고 있는 제어코멘드 Control A-2, Control B-2, Control C-2로 개별 변환할 수 있게 한다.

<78> 상술한 본 발명의 설명에서는 구체적인 실시 예에 관해 설명하였으나, 여러 가지 변형이 본 발명의 범위에서 벗어나지 않고 실시할 수 있다. 따라서 본 발명의 범위는 설명된 실시 예에 의하여 정할 것이 아니고 특허청구범위와 특허청구범위의 균등한 것에 의해 정해 져야 한다.

【발명의 효과】

<79> 상술한 바와 같이 본 발명은 통신 시스템에서 상위 어플리케이션 계층과 디바이스 드라이버 계층 사이에 DIA(Device Independent Access) 계층을 두어, 상위 어플리케이션은 디바이스 드라이버가 아닌 DIA를, 역시 디바이스 드라이버는 상위 어플리케이션이 아

닌 DIA의 표준 룰(rule)을 통해 상호 액세스하므로 직접적인 액세스를 피하게 한다. 이렇게 함으로써, 상위 어플리케이션 및 하위의 디바이스 드라이버가 개별화되고 특화되는 성향을 배제시키고 표준화 룰을 적용하여 개발의 요구 시간(need time)을 줄일 뿐 아니라 개발비용의 절감, 개발의 신뢰성을 크게 향상시킬 수 있다.

【특허청구범위】

【청구항 1】

디바이스 드라이버 제어 공통화 방법에 있어서,

디바이스 독립형 액세스(DIA)계층을 어플리케이션 계층과 디바이스 드라이버 계층에 사이에 위치시키며, 상기 DIA계층에 상위 어플리케이션 계층 및 디바이스 드라이버 계층에 대한 표준화 룰을 적용시키는 제1과정과,

상기 어플리케이션 계층 및 디바이스 드라이버 계층은 상기 DIA계층의 표준화된 룰을 통해 상기 디바이스 드라이버 계층 및 상위 어플리케이션 계층을 상호 액세스하는 제2과정으로 이루어짐을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 2】

제1항에 있어서, 상기 제2과정은,

상기 어플리케이션계층에 의해 해당 디바이스 드라이버에 대한 표준화된 공통 포맷의 코멘드를 상기 DIA계층으로 전송함에 따라 상기 DIA계층에 의해 상기 해당 디바이스 드라이버에 맞는 로컬 포맷으로 변환하여 상기 해당 디바이스 드라이버로 전달하는 단계와,

상기 해당 디바이스 드라이버에 의해 로컬 포맷의 응답을 상기 DIA계층으로 전송함에 따라 상기 DIA계층에 의해 상기 표준화된 공통 포맷의 응답으로 변환하여 상기 어플

리케이션 계층으로 전달하는 단계로 이루어짐을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 3】

디바이스 드라이버 제어 공통화 방법에 있어서,

디바이스 독립형 액세스(DIA)계층을 어플리케이션 계층과 디바이스 드라이버 계층에 사이에 위치시키는 과정과,

국제기구에서 기 정의된 표준화된 펑션블록의 펑션들중 디바이스 드라이버들 각각에서 사용 가능한 부분을 펑션 테이블로서 정의하는 과정과,

디바이스 초기화시 각 디바이스에 대한 표준화된 데이터 포맷의 디바이스 핸들러 식별자를 상기 디바이스 독립형 액세스 계층에 의해서 생성하여 상위의 어플리케이션으로 제공하는 과정과,

상기 상위의 어플리케이션이 상기 디바이스 핸들러 식별자를 가지고 소정의 디바이스를 호출함에 따라 상기 DIA계층이 상기 디바이스 핸들러 식별자를 이용하여 대응된 디바이스 드라이버의 펑션을 상기 펑션 테이블에서 찾아서 호출하는 과정으로 이루어짐을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 4】

제3항에 있어서, 상기 디바이스 핸들러 식별자는 DCB handlerId[x1.x2.x3](x는 비부호 정수)로서 표현되며, 상기 x1은 디바이스 식별자를 의미하는 레벨1의 값이고, 상기

x2는 해당 디바이스가 가지고 있는 논리적 혹은 물리적 그룹 번호를 의미하는 레벨2의 값이며, 상기 x3은 해당 디바이스 혹은 그룹이 가지고 있는 채널번호를 의미하는 채널값임을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 5】

제4항에 있어서, 상기 x1,x2,x3 값이 "0"이면, 해당 레벨 혹은 채널이 존재하지 않음을 의미하며, 디바이스 초기화시 x값은 "1"부터 순차적으로 증가를 함을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 6】

디바이스 드라이버 제어 공통화 방법에 있어서,

디바이스 독립형 액세스(DIA)계층을 어플리케이션 계층과 디바이스 드라이버 계층에 사이에 위치시키는 과정과,

상기 어플리케이션 계층으로부터의 디바이스 초기화 제어가 있으면 상기 DIA계층에 의해 상기 디바이스의 레벨1 초기화, 레벨2 초기화, 채널 초기화를 수행하여 상기 디바이스에 대한 표준화된 데이터 포맷의 디바이스 핸들러 식별자를 생성하는 과정과,

상기 디바이스 핸들러 식별자에 대응된, 표준화 룰을 수행하기 위한 엘리먼트들을 담는 디바이스 제어 블록을 상기 DIA계층이 동적 할당하는 과정과,

상기 DIA계층이 상기 생성된 디바이스 핸들러 식별자를 상기 어플리케이션 계층으로 제공하는 과정과,

상기 어플리케이션 계층이 상기 디바이스 핸들러 식별자를 가지고 소정의 디바이스를 상기 DIA계층을 통해서 호출하는 과정으로 이루어짐을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 7】

제6항에 있어서, 상기 디바이스 제어 블록의 엘리먼트들은, 표준화된 고유한 값을 가지는 코멘드 식별자와 대응 코멘드 펑션 포인터가 맵핑되어 있는 코멘드 제어 테이블의 위치를 알려주는 포인터 *pControlTable, 해당 기능의 존재 여부 및 위치를 알려주기 위한 디바이스 드라이버 제어 테이블이 있는 위치를 알려주는 포인터 *pDDCB, 다음 레벨을 지시하는 포인터 *pAnchor를 적어도 포함하고 있음을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 8】

제6항에 있어서, 상기 디바이스 제어 블록의 엘리먼트들은, 디바이스 초기화시 주어지는 초기화 프로파일의 위치를 알려주는 포인터 *pHandler, 디바이스 초기화시 사용되는 펑션이 있는 펑션 포인터 *fpIntivDevice, 채널을 열 때 사용되는 펑션이 있는 펑션 포인터 *fpOpenChannel, 채널을 닫을 때 사용되는 펑션이 있는 펑션 포인터 *fpCloseChannel, 열린 채널의 데이터를 읽을 때 사용되는 펑션 포인터 *fpRead, 열린 채널의 데이터를 쓸대 사용되는 펑션 포인터 *fpWrite, 디바이스 리셋시의 펑션 포인터 *fpReset, 표준화된 고유한 값을 가지는 코멘드 식별자와 대응 코멘드 펑션 포인터가 맵

핑되어 있는 코멘드 제어 테이블의 위치를 알려주는 포인터 *pControlTable, 해당 기능의 존재 여부 및 위치를 알려주기 위한 디바이스 드라이버 제어 테이블이 있는 위치를 알려주는 포인터 *pDDCB, 이벤트 테이블이 있는 위치를 알려주는 포인터 *pEventTable, 다음 레벨을 지시하는 포인터 *pAnchor로 구성됨을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 9】

제6항에 있어서, 상기 디바이스의 레벨1 초기화는, DCB handlerId[x1.x2.x3](x는 비부호 정수)로서 표현되는 디바이스 핸들러 식별자에서 레벨1 초기화되는 순서에 따라 디바이스들 각각에 대응된 디바이스 식별자 x1값을 고유한 값으로 부여하는 것임을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【청구항 10】

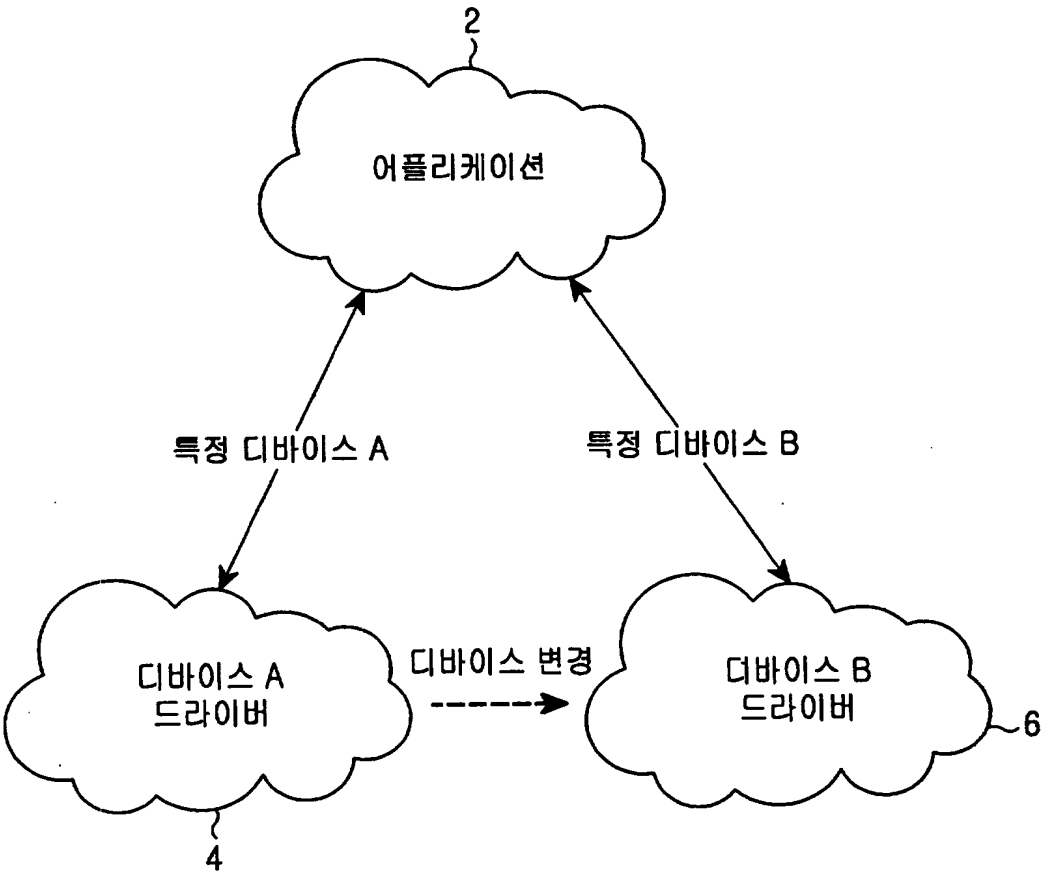
제9항에 있어서, 상기 디바이스의 레벨2 초기화는, 상기 레벨1 초기화 후에 수행되며 DCB handlerId[x1.x2.x3](x는 비부호 정수)로서 표현되는 디바이스 핸들러 식별자에서 디바이스들 각각에 대응된 논리적 혹은 물리적 그룹 개수를 참조하여 앵커들을 할당하고, 할당된 앵커들 각각에 대한 그룹값 x2 값을 각각의 고유한 값으로 부여하는 것임을 특징으로 하는 다바이스 드라이버 제어 공통화방법.

【청구항 11】

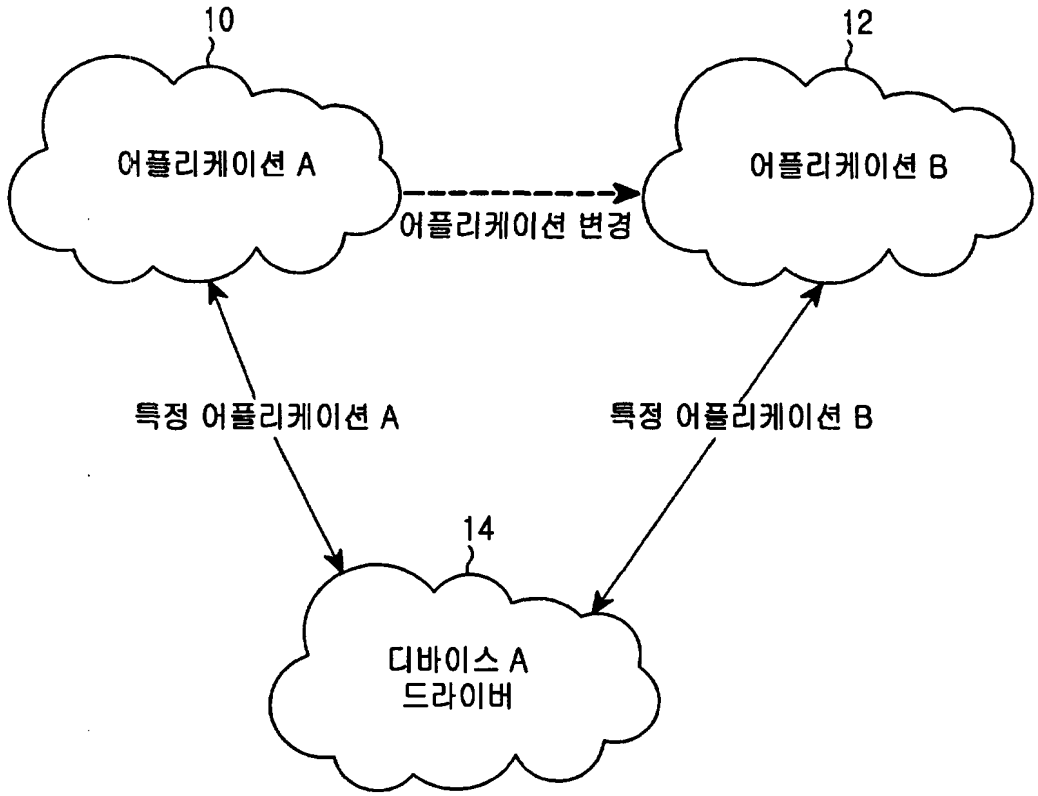
제10항에 있어서, 상기 디바이스의 레벨3 초기화는, DCB handlerId[x1.x2.x3](x는 비부호 정수)로서 표현되는 디바이스 핸들러 식별자에서 상기 디바이스나 상기 디바이스 내 그룹들이 가지고 있는 채널들 각각에 대해서 채널 오픈 순서에 따라 채널값 x3을 부여하는 것임을 특징으로 하는 디바이스 드라이버 제어 공통화 방법.

【도면】

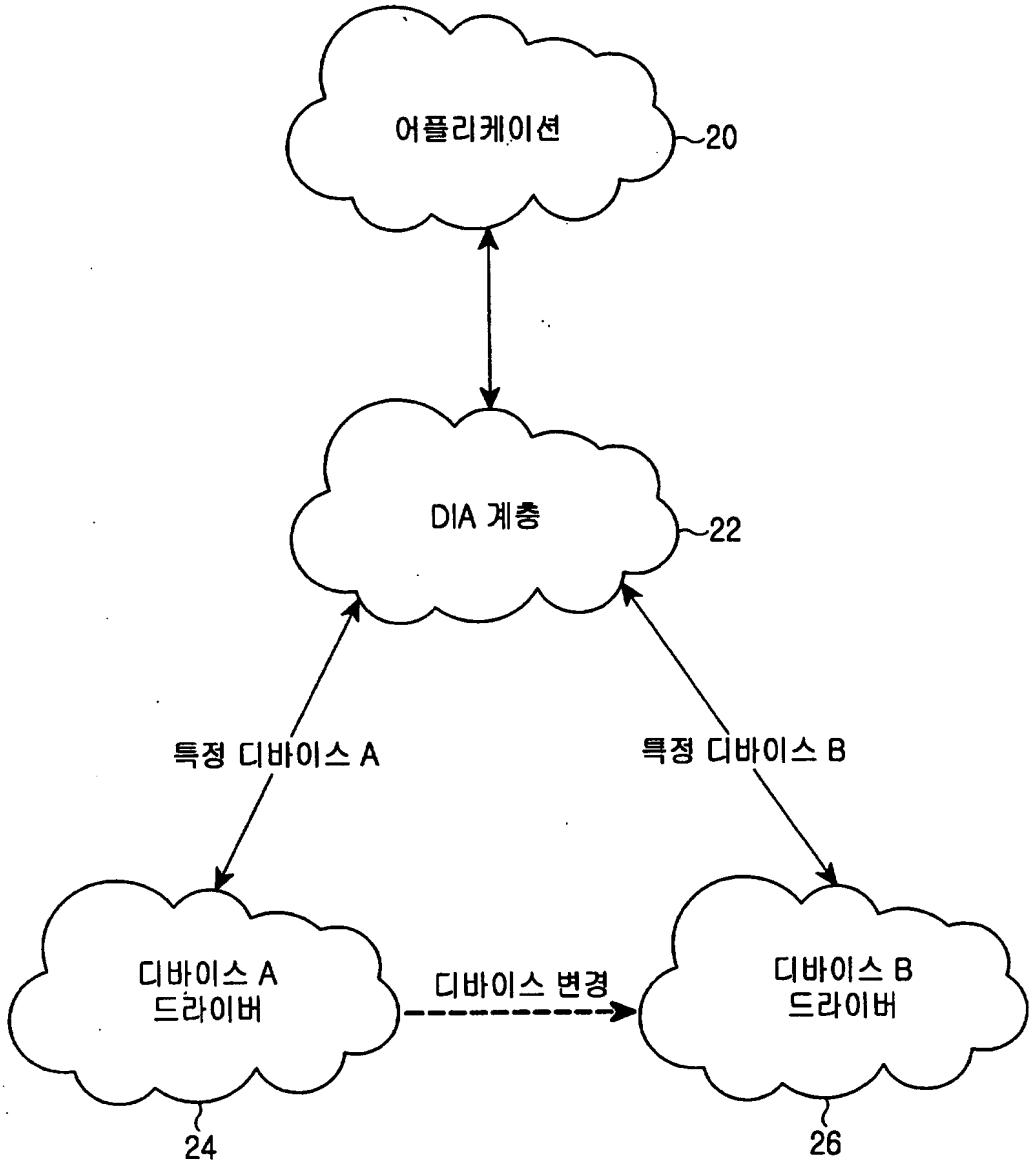
【도 1】



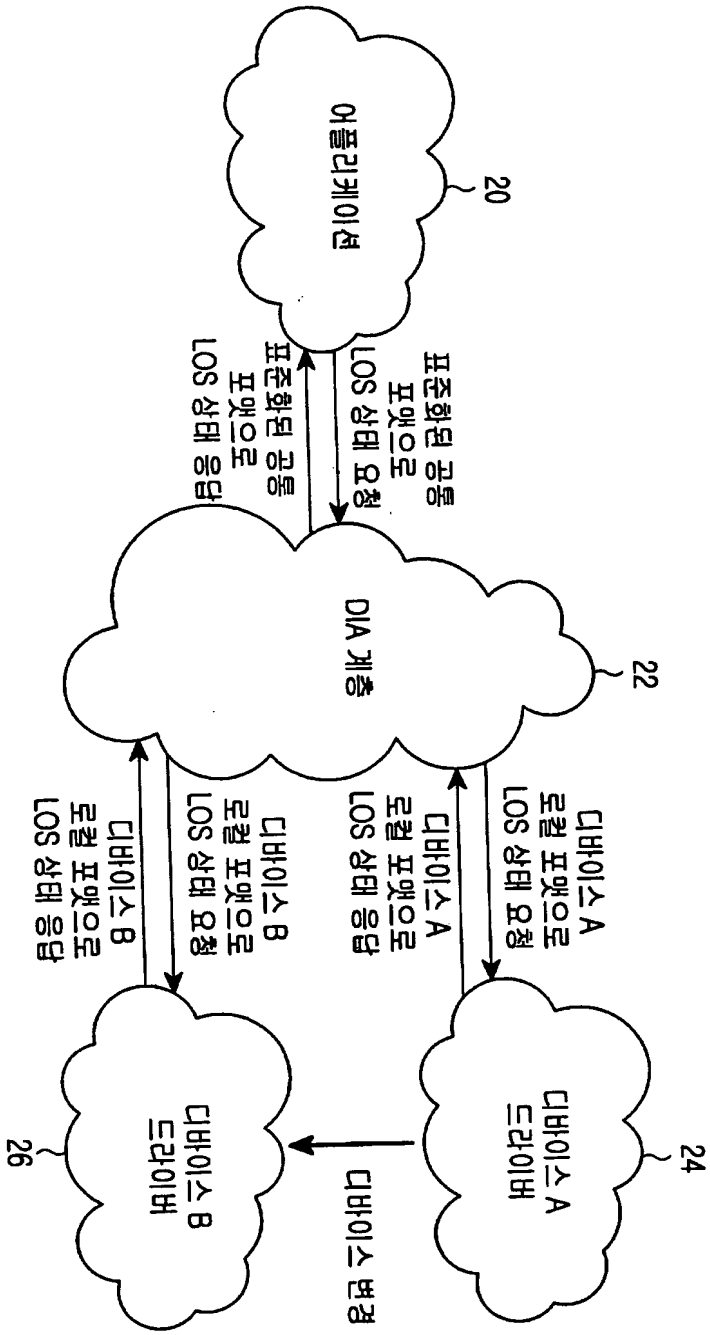
【도 2】


10
12
어플리케이션 A
어플리케이션 B
어플리케이션 변경
특정 어플리케이션 A
특정 어플리케이션 B
14
디바이스 A
드라이버

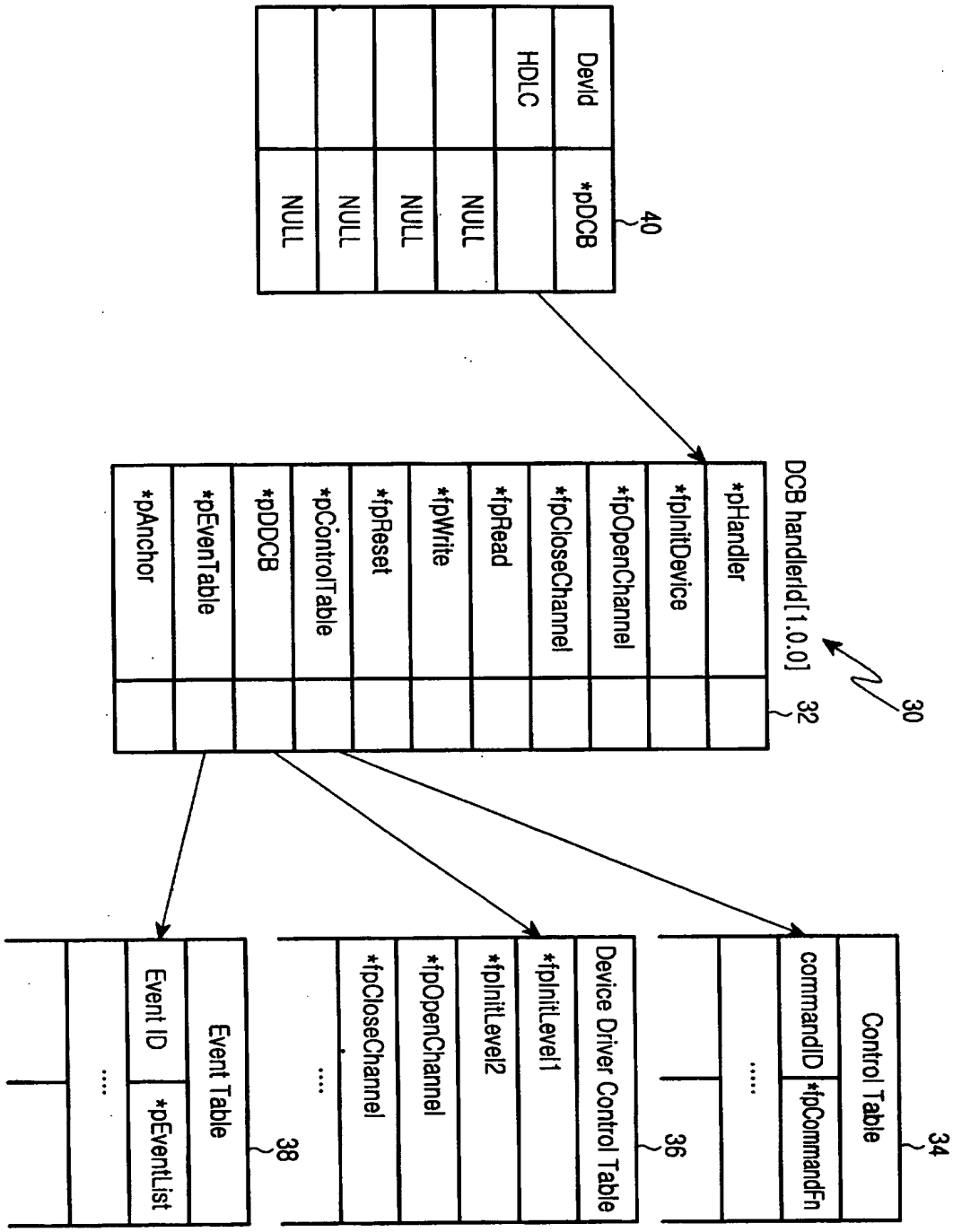
【도 3】


어플리케이션
20
DIA 계층
22
특정 디바이스 A
특정 디바이스 B
디바이스 A
드라이버
디바이스 변경
디바이스 B
드라이버
24
26

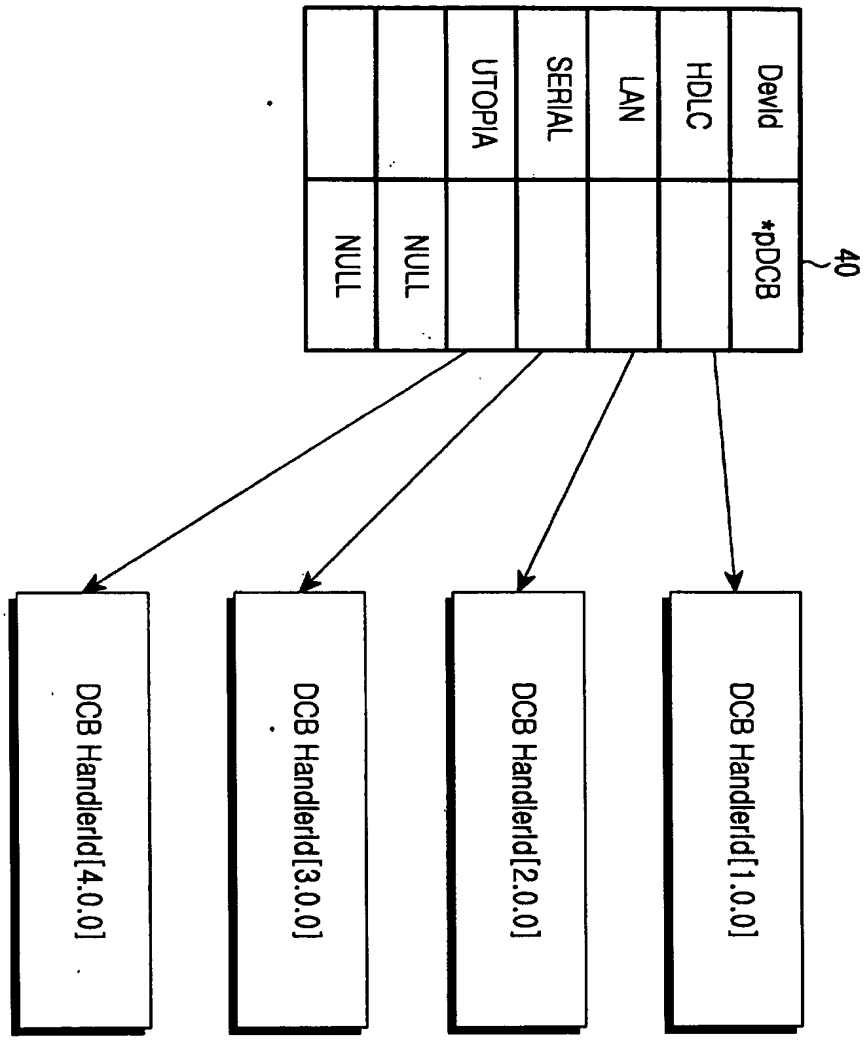
【도 4】


어플리케이션
20
표준화된 공통 포맷으로 LOS 상태 요청
표준화된 공통 포맷으로 LOS 상태 응답
DIA 계층
22
디바이스 A 로컬 포맷으로 LOS 상태 요청
디바이스 A 로컬 포맷으로 LOS 상태 응답
디바이스 B 로컬 포맷으로 LOS 상태 요청
디바이스 B 로컬 포맷으로 LOS 상태 응답
디바이스 A 드라이버
24
디바이스 B 드라이버
26
디바이스 변경

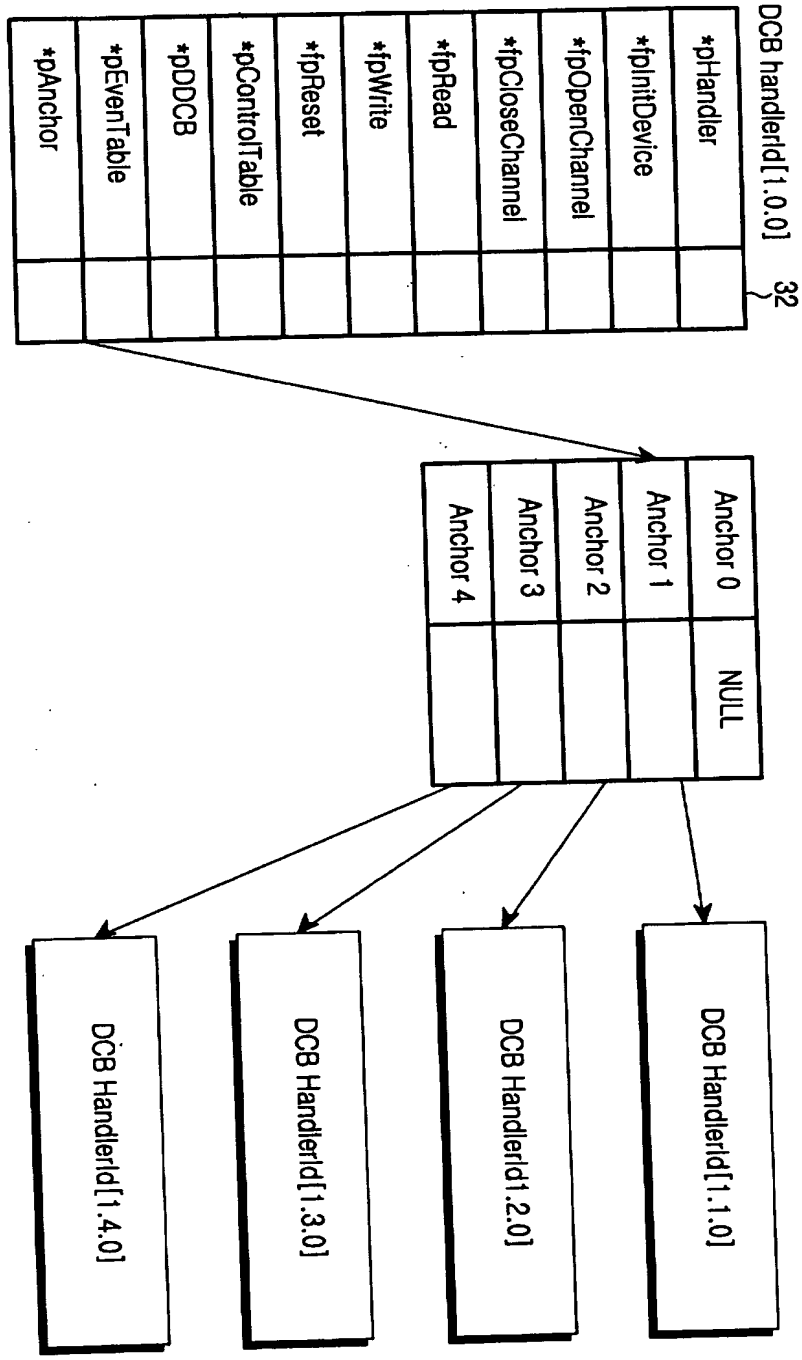
【도 5】


DevId
HDLC
NULL
NULL
NULL
NULL
*pDCB
40
DCB handlerId[1.0.0]
*pHandler
*fpInitDevice
*fpOpenChannel
*fpCloseChannel
*fpRead
*fpWrite
*fpReset
*pControlTable
*pDDCB
*pEvenTable
*pAnchor
32
30
Control Table
commandID
*fpCommandFn
.....
34
Device Driver Control Table
*fpInitLevel1
*fpInitLevel2
*fpOpenChannel
*fpCloseChannel
....
36
Event Table
Event ID
*pEventList
....
38

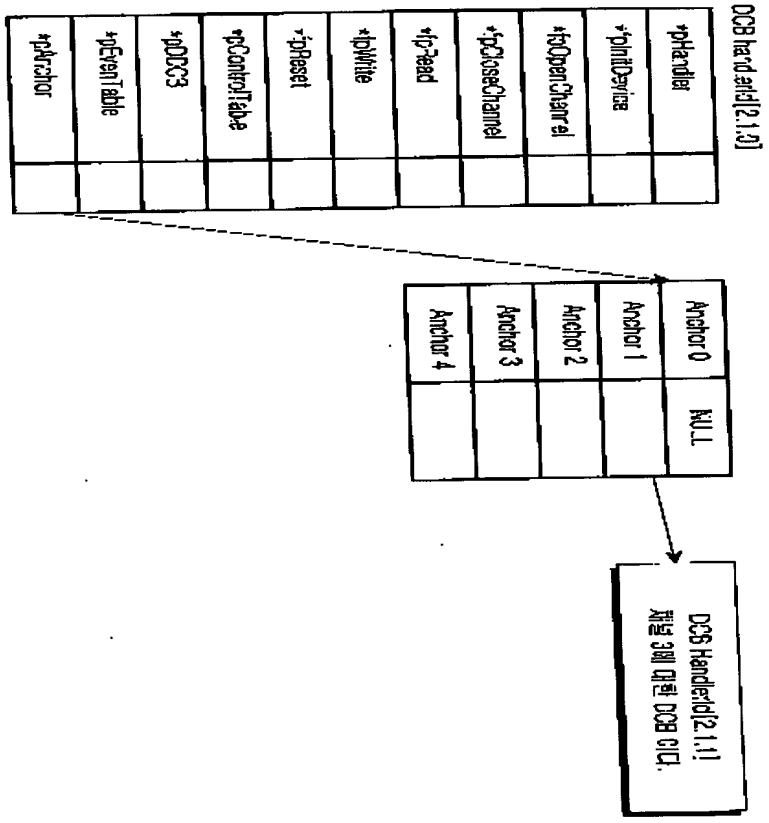
【도 6】


DevId
HDLC
LAN
SERIAL
UTOPIA
*pDCB
NULL
NULL
40
DCB HandlerId[1.0.0]
DCB HandlerId[2.0.0]
DCB HandlerId[3.0.0]
DCB HandlerId[4.0.0]

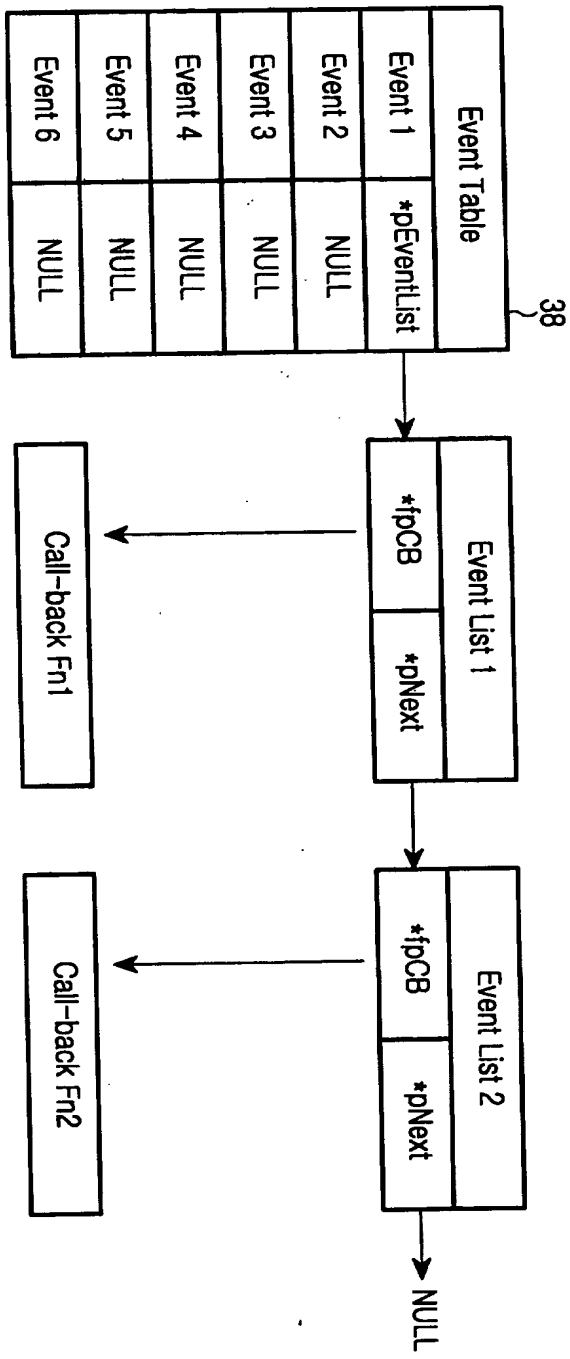
【도 7】


DCB handlerId[1.0.0]
*pHandler
*fpInitDevice
*fpOpenChannel
*fpCloseChannel
*fpRead
*fpWrite
*fpReset
*pControlTable
*pDDCB
*pEvenTable
*pAnchor
32
Anchor 0
Anchor 1
Anchor 2
Anchor 3
Anchor 4
NULL
DCB HandlerId[1.1.0]
DCB HandlerId[1.2.0]
DCB HandlerId[1.3.0]
DCB HandlerId[1.4.0]

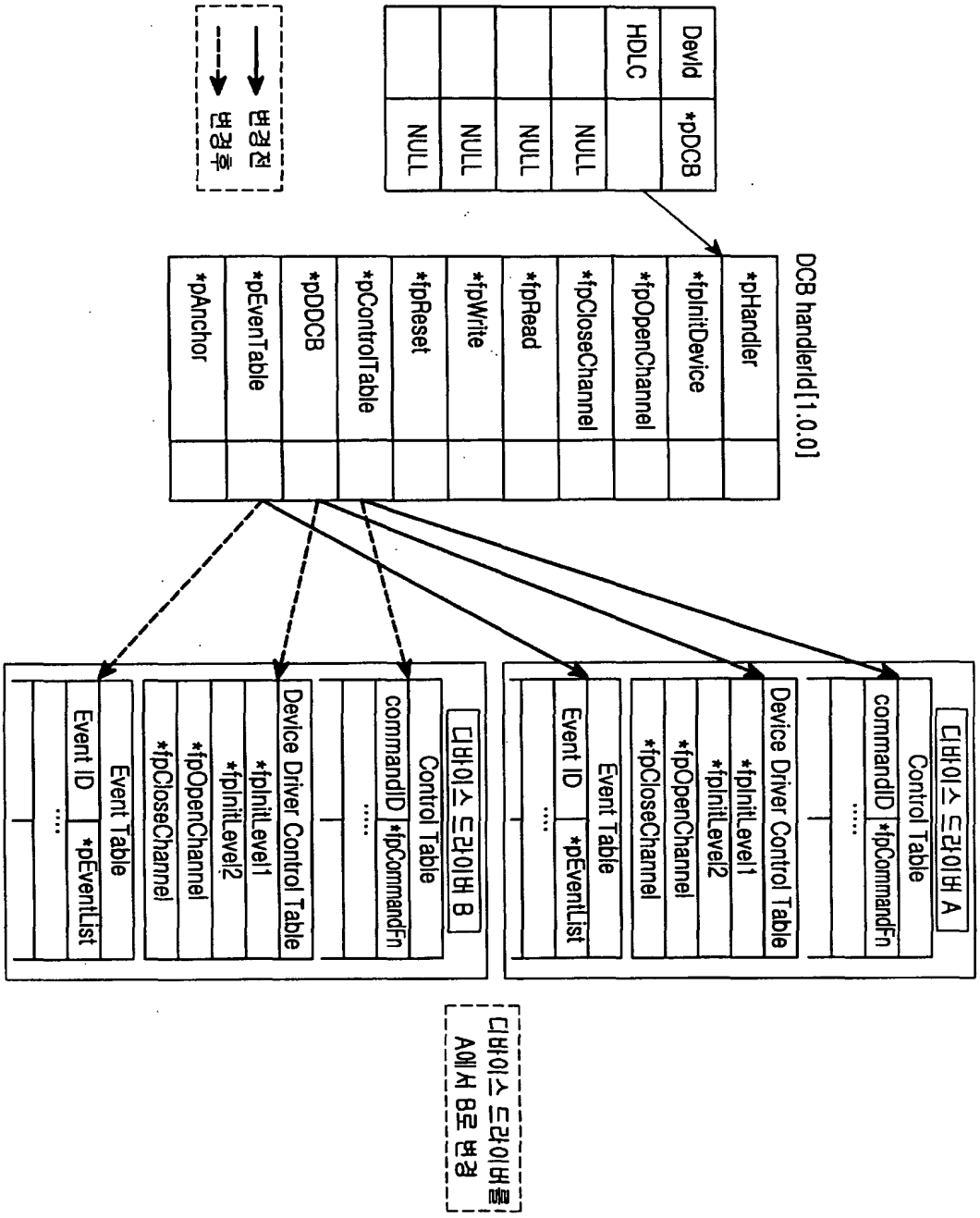
【도 8】


DCB hardwareId[2.1.3]
*pHandler
*pInitDevice
*fpOpenChannel
*fpCloseChannel
*fpRead
*fpWrite
*fpReset
*pControlTable
*pDCB3
*pEventTable
*pAnchor
Anchor 0
Anchor 1
Anchor 2
Anchor 3
Anchor 4
NULL
DCB HandlerId[2.1.1]
채널 3에 대한 DCB 입니다.

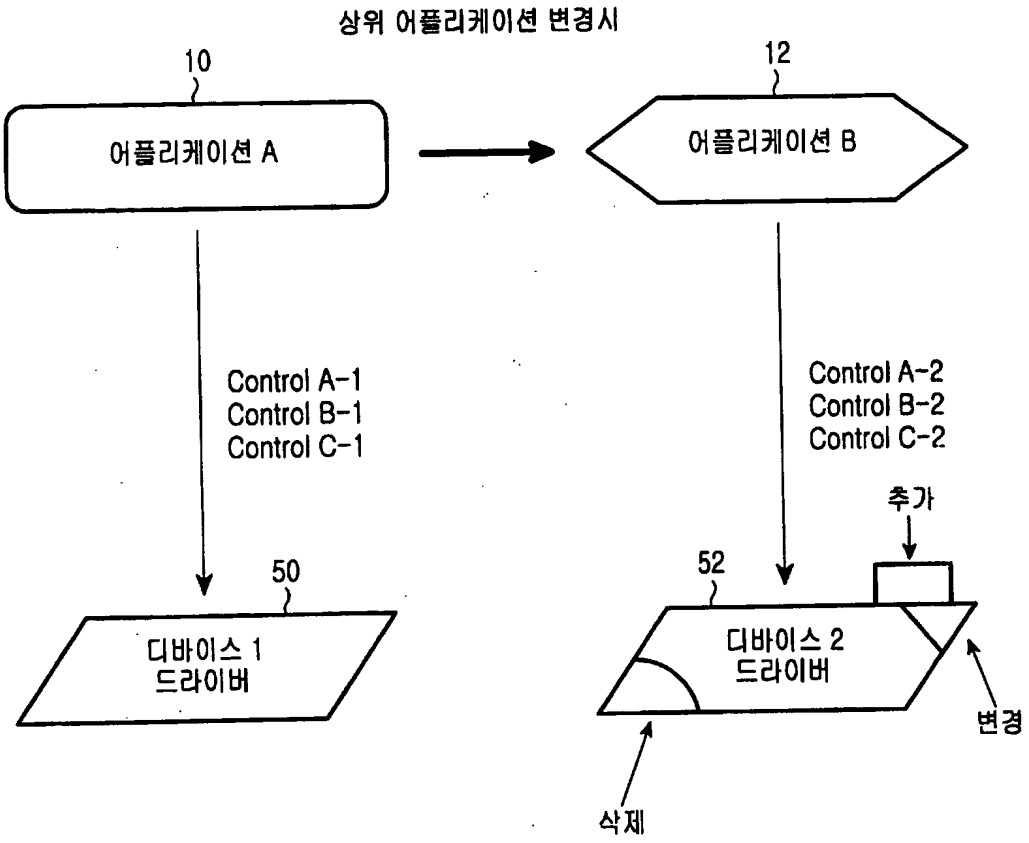
【도 9】


Event Table
Event 1
*pEventList
Event 2
NULL
Event 3
NULL
Event 4
NULL
Event 5
NULL
Event 6
NULL
38
*fpCB
*pNext
Event List 1
Call-back Fn1
*fpCB
*pNext
Event List 2
Call-back Fn2
NULL

【도 10】

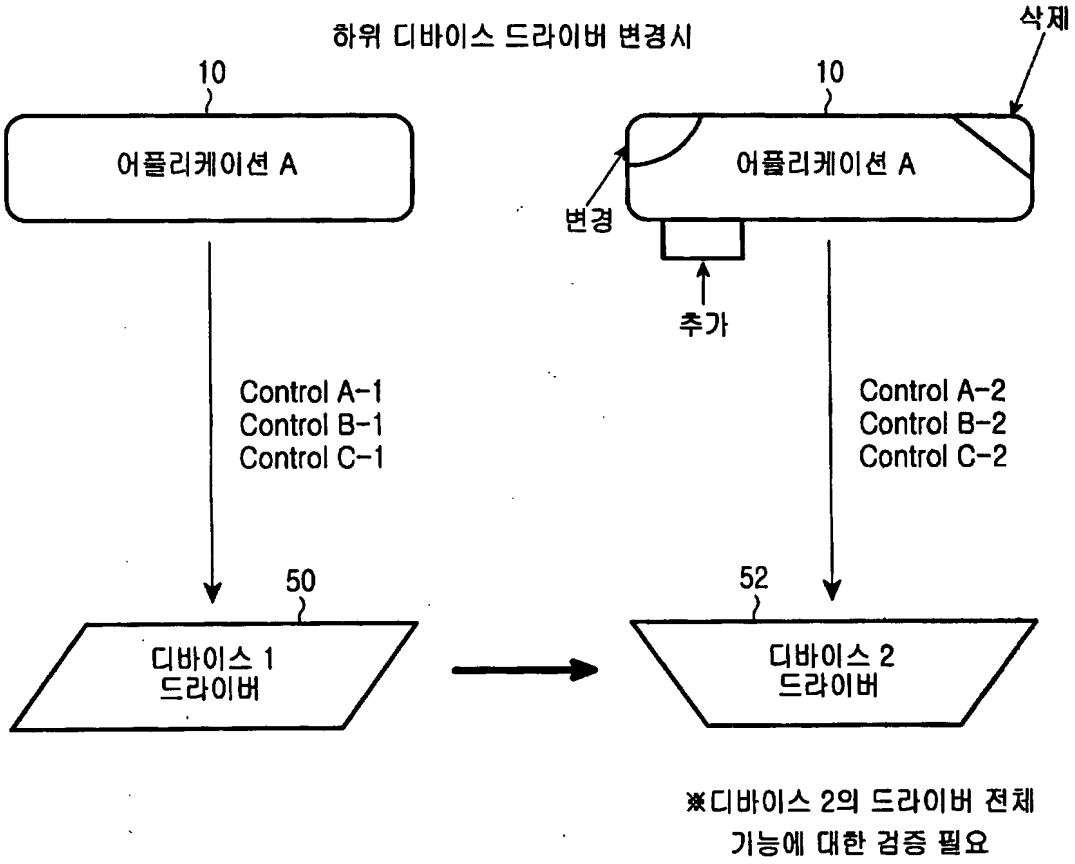

DevId
*pDCB
HDLC
NULL
변경전
변경후
DCB handlerId[1.0.0]
*pHandler
*fpInitDevice
*fpOpenChannel
*fpCloseChannel
*fpRead
*fpWrite
*fpReset
*pControlTable
*pDDCB
*pEventTable
*pAnchor
디바이스 드라이버 A
Control Table
commandID
*fpCommandFn
Device Driver Control Table
*fpInitLevel1
*fpInitLevel2
*fpOpenChannel
*fpCloseChannel
Event Table
Event ID
*pEventList
디바이스 드라이버 B
Device Driver Control Table
Event Table
디바이스 드라이버를
A에서 B로 변경

【도 11】

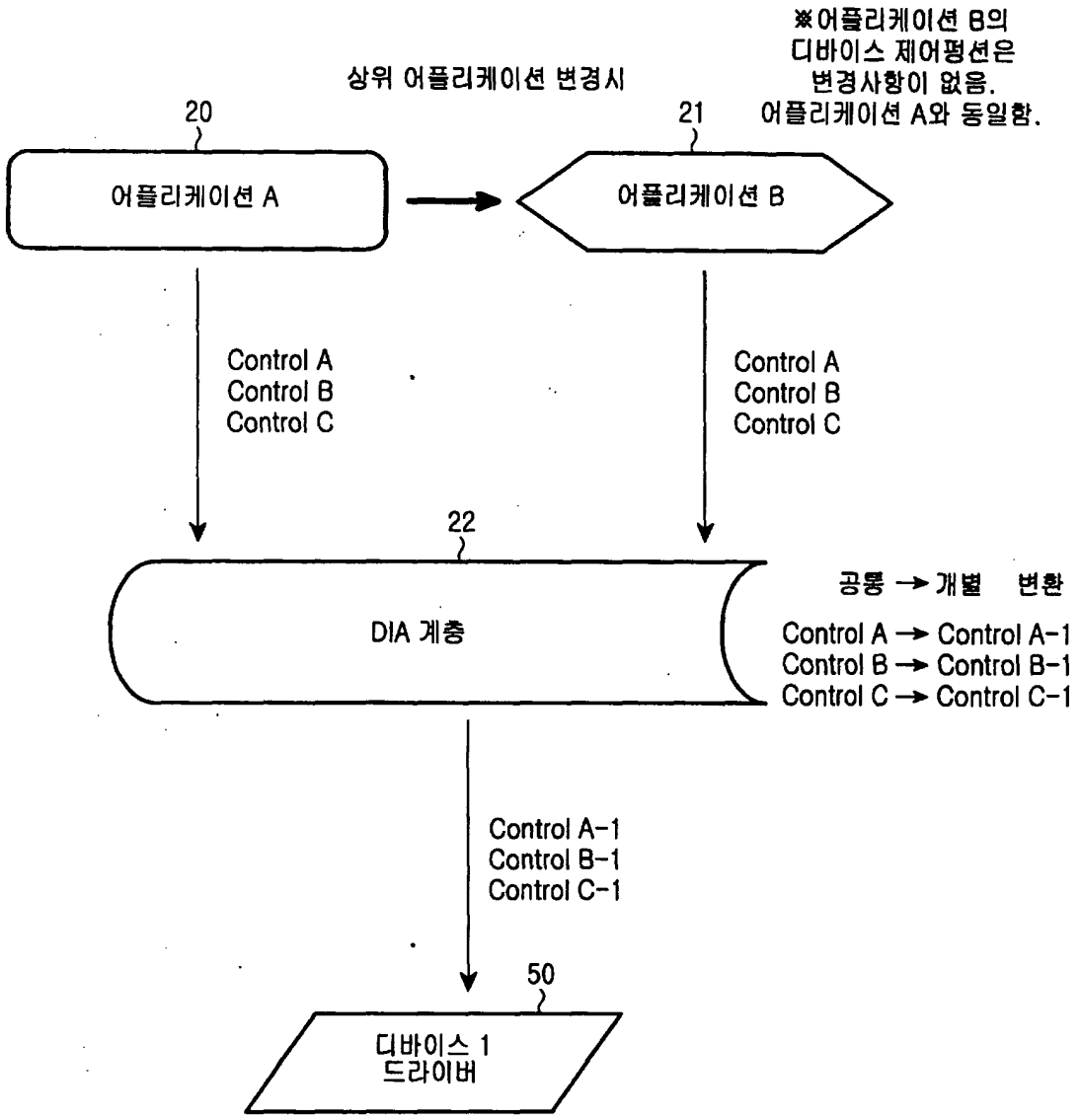


※삭제, 변경, 추가부분 검증 필요

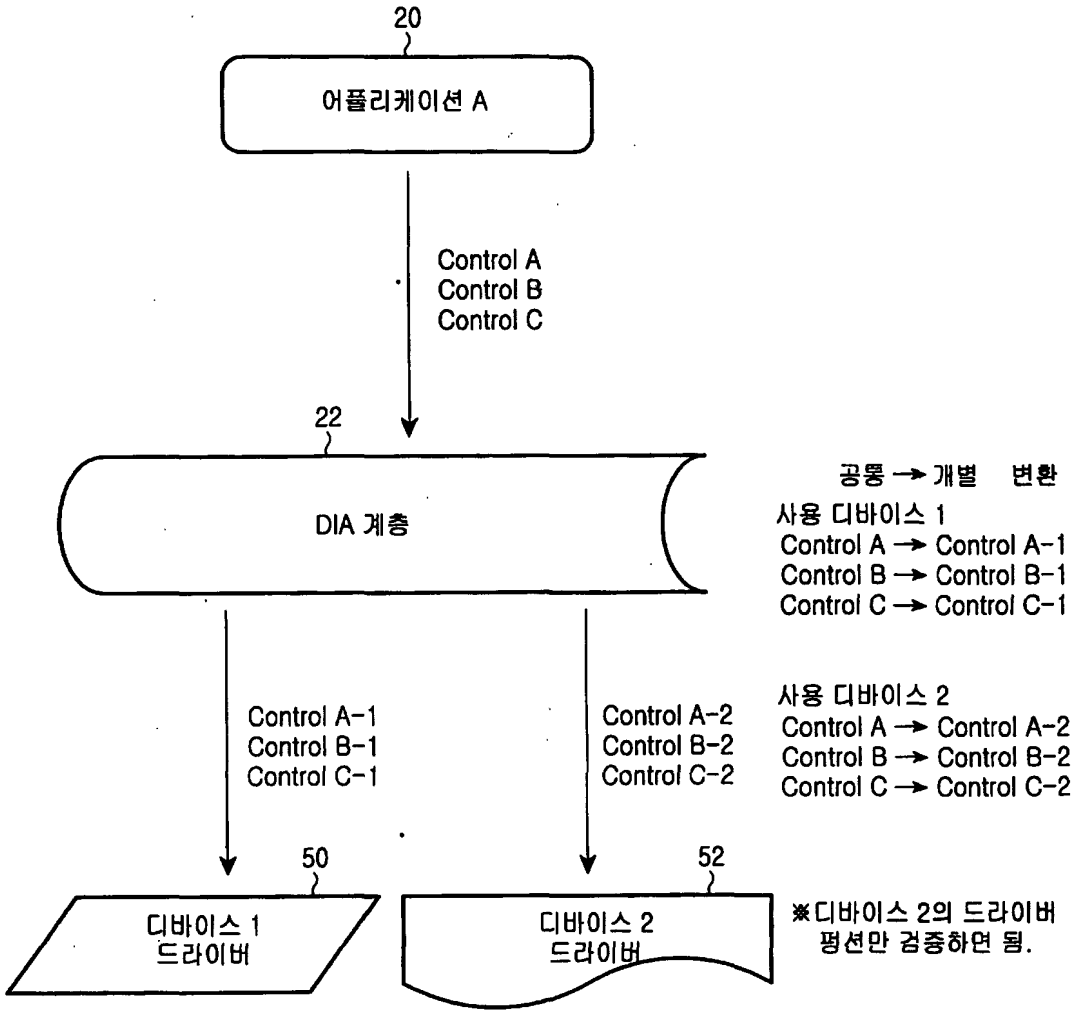
【도 12】


하위 디바이스 드라이버 변경시
삭제
10
어플리케이션 A
10
어플리케이션 A
변경
추가
Control A-1
Control B-1
Control C-1
Control A-2
Control B-2
Control C-2
50
디바이스 1
드라이버
52
디바이스 2
드라이버
※디바이스 2의 드라이버 전체
기능에 대한 검증 필요

【도 13】


※어플리케이션 B의
디바이스 제어펑션은
변경사항이 없음.
어플리케이션 A와 동일함.
상위 어플리케이션 변경시
20
21
어플리케이션 A
어플리케이션 B
Control A
Control B
Control C
Control A
Control B
Control C
22
공통 → 개별 변환
DIA 계층
Control A → Control A-1
Control B → Control B-1
Control C → Control C-1
Control A-1
Control B-1
Control C-1
50
디바이스 1
드라이버

【도 14】


20
어플리케이션 A
Control A
Control B
Control C
22
DIA 계층
공통 → 개별 변환
사용 디바이스 1
Control A → Control A-1
Control B → Control B-1
Control C → Control C-1
사용 디바이스 2
Control A → Control A-2
Control B → Control B-2
Control C → Control C-2
Control A-1
Control B-1
Control C-1
Control A-2
Control B-2
Control C-2
50
52
디바이스 1
드라이버
디바이스 2
드라이버
※디바이스 2의 드라이버
펑션만 검증하면 됨.